归档存储

最佳实践

最佳实践

Python SDK 简易使用示例

本节以示例的方式展示如何使用SDK高级接口进行开发。用户在阅读本节后，可模仿示例，并参考高级接口一节进行开发。

其中，方括号内的参数用户应根据实际需求进行替换。

- **创建 Vault**

```
from oas.oas_api import OASAPI
from oas.ease.vault import Vault

# 创建 OASAPI 对象
api = OASAPI('[Server Host]', '[Access Key ID]', '[Access Key Secret]')

# 创建 Vault
vault = Vault.create_vault(api, '[Vault Name]')
```

- **查找Vault**

```
# 创建 OASAPI 对象
api = OASAPI('[Server Host]', '[Access Key ID]', '[Access Key Secret]')

# 根据名称获取 Vault
vault = Vault.get_vault_by_name(api, '[Vault Name]')

# 根据 ID 获取 Vault
vault = Vault.get_vault_by_id(api, '[Vault ID]')
```

- **上传文件**

```
archive_id = vault.upload_archive('[File Path]')
```

- **删除 Archive**

```
vault.delete_archive('[Archive ID]')
```

- **续传 Multipart Upload 任务**

```
uploader = vault.recover_uploader('[Upload ID]')
uploader.resume('[File Path]')
```

- **获取 Archive 列表**

```
job = vault.retrieve_inventory()
job.download_to_file('[File Path]')
```

- **下载 Archive**

```
job = vault.retrieve_archive('[Archive ID]')
job.download_to_file('[File Path]')
```

- **从OSS上转储Object到OAS**

```
job = vault.pull_from_oss(conf.osshost, conf.bucket, conf.object, "test desc")
```

- **从OAS上转储Archive到OSS**

```
job = vault.push_to_oss(archive_id, conf.osshost, conf.bucket, archive_id, "test desc")
```

Python SDK 完整使用演示代码

下面函数中test_single_archive_upload提供单一文件archive上传；

函数test_multi_upload() 使用sdk低级接口实现分段上传；

函数test_uploader()使用sdk高级接口实数据上传（当数据大于64MB时，会自动选择分段上传）；

函数test_vault_retrieve()实现vault信息查询；

函数test_download_archive(archive_id)实现archive下载；

函数test_delete_archive(archive_id)实现archive删除。

函数test_pull_from_oss（ ）实现从OSS直接转储到OAS

函数test_push_to_oss() 实现从OAS直接转储到OSS

```
import random
import time
import logging

import logging.handlers
from oas.oas_api import OASAPI
from oas.ease.api import APIProxy
```

```
from oas.ease.exceptions import *
from oas.ease.response import *
from oas.ease.utils import *
from oas.ease.vault import *
from oas.ease.uploader import *
from oas.ease.job import *

import os

LOG_FILE="test.log"
handler = logging.handlers.RotatingFileHandler(LOG_FILE, maxBytes = 1024*1024, backupCount = 5)
fmt = '%(asctime)s - %(filename)s:%(lineno)s - %(name)s - %(message)s'
formatter = logging.Formatter(fmt)
handler.setFormatter(formatter)
log.addHandler(handler)
log.setLevel(logging.DEBUG)


class TestConf(object):

def __init__(self):
self.host = 'oas域名'
self.accesskey_id = "您的access key Id"
self.accesskey_secret = "您的access key secret"
self.vault_name = "normal"
self.vault_name_test = "test"

self.osshost = "您要转储的oss域名"
self.bucket = "您要转储的bucket"
self.object = "您要转储的Object"

conf = TestConf()


class TestDemo():
_MEGABYTE = 1024 * 1024


def __init__(self):
self.api = OASAPI(conf.host, conf.accesskey_id, conf.accesskey_secret)
self.api_proxy = APIProxy(self.api)
self.vault_name = conf.vault_name
self.vault_name_test = conf.vault_name_test

self.file_big = "random100M.bin"
self.file_big_size = 200*self._MEGABYTE

with open(self.file_big, 'wb+') as f:
self.write_random_data(f, self.file_big_size)

self.file_small = "random30M.bin"
self.file_small_size = 30*self._MEGABYTE

with open(self.file_small, 'wb+') as f:
self.write_random_data(f, self.file_small_size)
```

```
def write_random_data(self, file, size):
remaining = size
while remaining > 0:
n = min(remaining, 1024)
random_data = os.urandom(n)
file.write(random_data)
remaining = remaining - n

n = min(remaining, 1024 * random.randint(256, 1024))
file.write('\0' * n)
remaining = remaining - n


def test_single_archive_upload(self):
print '''
############################################################
Using the low level api method to upload a small archive
############################################################'''
res = self.api_proxy.create_vault(self.vault_name)
vault_id = res['x-oas-vault-id']

etag = compute_etag_from_file(self.file_small)
tree_etag = compute_tree_etag_from_file(self.file_small)
with open(self.file_small, 'rb') as f:
content = f.read()
res = self.api_proxy.post_archive(vault_id, content, etag, tree_etag)
archive_id = res['x-oas-archive-id']
print "archive_id",archive_id

#test the multipart upload by the proxy api, and this is low level api
def test_multi_upload(self):
print '''\n\n\n
############################################################
Using the low level api to invoke the multipart api and implement the archive uplod
############################################################'''
res = self.api_proxy.create_vault(self.vault_name)
vault_id = res['x-oas-vault-id']

part_size = 1024 * 1024 * 64


etag_array= []
tree_etag_array= []
offset = 0

cur_part_num=0
cur_start_pos = 0
cur_end_pos = 0
print "1. comput the etag,tree-etag"
print "1.1 comput the etag , tree_etag of part"
while True:
tmpsize = part_size
if(cur_start_pos + tmpsize > self.file_big_size):
tmpsize = self.file_big_size - cur_start_pos;
cur_end_pos += tmpsize -1
```

```
etag_array.append(compute_etag_from_file(self.file_big, cur_start_pos, tmpsize) )
tree_etag_array.append(compute_tree_etag_from_file(self.file_big, cur_start_pos, tmpsize))

cur_start_pos += tmpsize
cur_part_num += 1
if(cur_start_pos >= self.file_big_size-1):
break;

print "1.2 comput the total tree_etag of the archive"
tree_etag_total = compute_combine_tree_etag_from_list( tree_etag_array) ;

print "2.1 init the upload task, and get the uploadId"
res = self.api_proxy.create_multipart_upload(vault_id, part_size)
upload_id = res['x-oas-multipart-upload-id']
print "upload_id",upload_id

f = open(self.file_big, 'rb')

cur_part_num = 0
cur_start_pos = 0
cur_end_pos = 0
while True:
tmpsize = part_size
if(cur_start_pos + tmpsize > self.file_big_size):
tmpsize = self.file_big_size - cur_start_pos;

cur_end_pos += tmpsize
print "2.2 upload every part to oas server, and the etag of the part will be used. current part is:", cur_part_num+1
self.api_proxy.post_multipart_from_reader(vault_id, upload_id, f, tmpsize,('%d-%d' %(cur_start_pos, cur_end_pos-1)),
etag_array[cur_part_num],tree_etag_array[cur_part_num])

cur_start_pos += tmpsize
cur_part_num += 1
if(cur_end_pos>= self.file_big_size -1):
break;

print "2.3 complete the multipart task, and the total etag will be used"
res = self.api_proxy.complete_multipart_upload(
vault_id, upload_id, self.file_big_size, tree_etag_total)
print "output the archive id"
archive_id = res['x-oas-archive-id']
print "archive_id",archive_id
return archive_id

#test the uploader to upload big file, and this is high level api
def test_uploader(self):
print '''\n\n\n
############################################################
Using the High level api to invoke the multipart api
############################################################'''
vault = Vault.create_vault(self.api, self.vault_name)

print "initial a uploadId"
uploader = vault.initiate_uploader(self.file_big)
uploader_id = uploader.id
```

```
print "uploader_id",uploader_id

print "start the multipart"
archive_id = uploader.start()

print "finish the upload, and output the archive_id"
print "archive_id",archive_id
return archive_id


#to inquire the archive list of vault
def test_vault_retrieve(self):
print '''\n\n\n
############################################################
Retrieve the vault info, and inquire the archive list of the vault
############################################################'''
vault = Vault.create_vault(self.api, self.vault_name)
job = vault.retrieve_inventory()
job.download_to_file(job.id)

#to test download archive
def test_download_archive(self, archive_id):
print '''\n\n\n
############################################################
Submit the archive job and download the job to local
############################################################'''
vault = Vault.create_vault(self.api, self.vault_name)
job = vault.retrieve_archive(archive_id)
job.update_status()
if not job.completed:
job.download_to_file(file_path = job.id, block = True)

#test delete archive
def test_delete_archive(self, archive_id):
print '''\n\n\n
############################################################
Delete the archive
############################################################'''
vault = Vault.create_vault(self.api, self.vault_name)
vault.delete_archive(archive_id)

def test_pull_from_oss(self):
print '''\n\n\n
############################################################
submit a pull-from-oss job and OAS will finish the pull of OSS object to OAS
############################################################'''
vault = Vault.create_vault(self.api, self.vault_name)
# create vault
job = vault.pull_from_oss(conf.osshost, conf.bucket, conf.object, "test desc")
job._check_status(block=True)

# delete archive
print "bucket:%s object:%s finish pull from oss,\n archiveId:%s" % (conf.bucket, conf.object, job.archive_id)


def test_push_to_oss(self):
```

```
print '''\n\n\n
############################################################
submit a push-to-oss job and OAS will finish the push of OAS archive to OSS
############################################################'''
vault = Vault.create_vault(self.api, self.vault_name)
archive_id = vault.upload_archive(self.file_big)
job = vault.push_to_oss(archive_id, conf.osshost, conf.bucket, archive_id, "test desc")
job._check_status(block=True)
print archive_id + " finish push to oss"

if __name__ == '__main__':
t = TestDemo()

t.test_single_archive_upload()
t.test_multi_upload();

archive_id = t.test_uploader();

t.test_vault_retrieve();
t.test_download_archive(archive_id)
t.test_delete_archive(archive_id)

t.test_pull_from_oss()
t.test_push_to_oss()
```

AliyunOASClient和ArchiveManager

AliyunOASClient和ArchiveManager分别是低阶接口、高级接口的入口对象，在每个功能中都会使用到它们，为避免实例程序的冗长，将AliyunOASClient和ArchiveManager的创建在这里单独介绍。

> 注：在本文档中出现的 aliyunOASClient 均是 AliyunOASClient 实例，archiveManager 均是 ArchiveManager实例

AliyunOASClient实例化

```
// AccessId和AccessKey配置
ServiceCredentials credentials = new ServiceCredentials(
    "[yourAccessId]", "[yourAccessKey]");
// 配置客户端设置
ClientConfiguration clientConf = new ClientConfiguration();
// 指定要连接的服务地址
ServiceHost serviceHost = new ServiceHost("http://cn-hangzhou.oas.aliyuncs.com");

// 用OAS工厂获得AliyunOASClient对象
AliyunOASClient aliyunOASClient = OASFactory.aliyunOASClientFactory(
    serviceHost, credentials, clientConf);
```

```
// 或者像下面这样，通过ServiceCredentials和url直接创建AliyunOASClient对象
AliyunOASClient aliyunOASClient = OASFactory.aliyunOASClientFactory(
    credentials, "http://cn-hangzhou.oas.aliyuncs.com");
```

ArchiveManager实例化

```
// 使用默认配置
ServiceCredentials credentials = new ServiceCredentials("[yourAccessKeyID]", "[yourAccessKeySecret]");

ArchiveManager manager = OASFactory.archiveManagerFactory(
    credentials, "http://cn-hangzhou.oas.aliyuncs.com");


// 自定义配置ClientConfiguration
ServiceCredentials credentials = new ServiceCredentials("[yourAccessKeyID]", "[yourAccessKeySecret]");
// 指定要连接的服务地址
ServiceHost serviceHost = new ServiceHost("http://cn-hangzhou.oas.aliyuncs.com");
// 配置客户端设置
ClientConfiguration clientConf = new ClientConfiguration();

ArchiveManager archiveManager = new ArchiveManager(serviceHost, credentials, clientConf);


// 使用AliyunOASClient初始化
AliyunOASClient aliyunOASClient = OASFactory.aliyunOASClientFactory(
    credentials, clientConf).withLogger();
ArchiveManager archiveManager = OASFactory.archiveManagerFactory(aliyunOASClient);
```

Vault操作

创建Vault

- 如果指定名称的Vault不存在，则创建Vault；
- 如果指定名称的Vault已存在，不执行创建动作，返回Vault的信息。

```
// vaultName必须满足已下2个条件：
// 1. 总长度在3~63之间（包括3和63）；
// 2. 只允许包含以下字符：
//    0-9(数字),
//    a-z(小写英文字母),
//    -(短横线),
//    _(下划线)
// 其中 短横线 和 下划线 不能作为vaultName的开头和结尾；
String vaultName = "oas_demo";

// 发送创建vault请求
// 创建Vault的名称，用CreateVaultRequest来指定
CreateVaultRequest createRequest = new CreateVaultRequest().
    withVaultName(vaultName);
```

```
// 发起创建Vault的请求
// 如果有同名的vault存在，OAS会返回已有的vault的vaultId，而不会执行创建动作
try {
    CreateVaultResult result = aliyunOASClient.createVault(createRequest);
    logger.info("Vault created vaultId={}", result.getVaultId());
    logger.info("Vault created vaultLocation={}", result.getLocation());
} catch (OASClientException e) {
    logger.error("OASClientException Occured:", e);
} catch (OASServerException e) {
    logger.error("OASServerException Occured:", e);
}
```

删除Vault

```
// 删除vault是高危动作，因此只能通过vaultId来删除
DeleteVaultRequest deleteVaultRequest = new DeleteVaultRequest().withVaultId(yourVaultId);

try {
    OASResult result = aliyunOASClient.deleteVault(deleteVaultRequest);
    logger.info("Delete Success! {} {}", result.getRequestId(), result.getDate());
} catch (OASClientException e) {
    logger.error("OASClientException Occured:", e);
} catch (OASServerException e) {
    logger.error("OASServerException Occured:", e);
}
```

上传Archive

快速上传

- ArchiveManager的upload接口会根据文件大小判断是直接上传还是用Multipart上传，无须用户选择

关于高级接口ArchiveManager的详细文档，参考 OAS Java SDK高级接口文档 。

```
// java sdk 会帮助用户完成 vaultName 到 vaultId 的转换
// 由此避免了用户对一长串vaultId的记忆负担
String yourVaultName = "oas_demo";
File file = new File("oas_demo_data/random10M.bin");
try {
    UploadResult uploadResult = archiveManager.upload(yourVaultName, file);
    logger.info("File {} uploaded complete. ArchiveId={},md5={},treeEtag={}",
        file.getAbsolutePath(), uploadResult.getArchiveId(),
        uploadResult.getHashValue(),
        uploadResult.getHashTreeValue());
} catch (OASClientException e) {
    logger.error("OASClientException Occured:", e);
} catch (OASServerException e) {
    logger.error("OASServerException Occured:", e);
```

```
}
```

根据UploadId上传

```
File file = new File("oas_demo_data/random120M-2.bin");
//获得uploadId
//文件大小必须大于100MB，否则会抛异常提示用普通上传接口进行上传
String uploadId = archiveManager.initiateMultipartUpload(vaultName, file, "Hello OAS!");
//如果是已有的uploadId，直接使用之前获取过的uploadId
//String uploadId = "[yourUploadId]";
System.out.println("Get uploadId=" + uploadId);
UploadResult uploadResult = archiveManager.uploadWithUploadId(vaultName, file, uploadId);
System.out.println("Archive ID=" + uploadResult.getArchiveId());
```

任务执行

下载任务是通过提交任务、下载任务输出的异步过程执行的，任务的详细介绍，参考 OAS Java SDK高级接口文档。

提交提档任务

```
String yourVaultName = "oas_demo";
String yourArchiveId =
"6086C62DBFFD4A68BAEE9D1B680EC77C8FE6AE617FF47BCD3DB3CB71AFAE29B33431AED14D06D9C4AD251F17C173F3CD";
JobMonitor jobMonitor = archiveManager.downloadAsync(yourVaultName, yourArchiveId);
```

执行Inventory

```
JobMonitor jobMonitor = archiveManager.downloadInventoryAsync("oas_demo");
```

下载Job输出

```
archiveManager.downloadJobOutput(yourVaultName, jobId,
    new File("oas_demo_data/mySaveFile.bin"));
```

配置ProcessListener

以上传Archive为例：

```
ServiceCredentials credentials = new ServiceCredentials(
    DemoConstants.ACCESS_ID,
```

```
    DemoConstants.ACCESS_KEY);

// 通过工厂类获得archiveManager接口
ArchiveManager archiveManager = OASFactory.archiveManagerFactory(credentials);
File file = new File("[pathToYourFile]");

final BaseTransfer<UploadResult> bt = archiveManager.uploadAsync(
    "[yourVaultName]", file);

//设置最大并发数，默认为3，最大为10
bt.setNumConcurrence(5);

bt.addProgressListener(new ProgressListener() {

    @Override
    public void onStart(String id) {
        // 任务开始时调用，其中id为Multipart Upload任务ID，对于一般上传任务id为空
        System.out.println("Start! Upload ID: " + id);
    }

    @Override
    public boolean onError(Range range, Throwable t) {
        // 出错时调用，range是出错的字节范围，t是相应的错误
        // 当返回true时，BaseTransfer会进行重试，false则放弃
        System.out.println("ERROR!!!");
        return false;
    }

    @Override
    public void onCompleted() {
        // 任务完成时调用
        System.out.println("Upload complete");
    }

    @Override
    public void onProgressed(long current, long total) {
        // 上传进度，total为文件字节大小，current为当前已上传字节数
        System.out.println("Progress: " + current + " / " + total);
    }
});

bt.start();

final Timer timer = new Timer();
timer.scheduleAtFixedRate(new TimerTask() {

    @Override
    public void run() {
        System.out.println("Running time: " + bt.getRunningTime() + " seconds");
        System.out.println("Completed size: " + bt.getSizeCompleted() + " bytes");
        System.out.println("Total size: " + bt.getSizeTotal() + " bytes");
        System.out.println("Average speed: " + bt.getAverageSpeedInBytesPerSecond()
                        + " B/s");
        if (bt.isComplete()) {
            timer.cancel();
            synchronized (bt) {
```

```
                bt.notify();
            }
        }
    }

}, 0, 1000);

synchronized (bt) {
    try {
        bt.wait();
    } catch (InterruptedException e) {
        logger.error("", e);
    }
}

// 任务结束
System.out.println("============================");
System.out.println("Running time: " + bt.getRunningTime() + " seconds");
System.out.println("Completed size: " + bt.getSizeCompleted() + " bytes");
System.out.println("Total size: " + bt.getSizeTotal() + " bytes");
System.out.println("Average speed: " + bt.getAverageSpeedInBytesPerSecond() + " B/s");
UploadResult uploadResult = bt.getResult();
System.out.println("Archive ID: " + uploadResult.getArchiveId());
System.out.println("Hash Value: " + uploadResult.getHashValue());
```

adup备份实践

简介

adup是基于开源备份工具duplicity，使用python语言实现，支持文件全量增量备份，支持小文件打包合并，提供了阿里云归档存储的备份工具。duplicity本身是一个多功能本地和远程的备份软件，支持多种后端存储介质如ftp、ssh和各类云存储。

特性

- 简单易用的命令行使用方式
- 使用rsync对变化数据进行增量备份，提高带宽和存储的高效利用
- 使用标准文件打包压缩格式GNU-tar，能够对增量数据进行追加
- 多种远程存储方式的支持，阿里云归档存储，AWS等

安装

debian发行版本

```
sudo bash deploy_debian.sh
```

redhat/centos发行版本

```
sudo bash deploy_rh.sh
```

安装完成以后，备份脚本dt-oas-backup.sh 存放在/usr/local/bin/下。

使用步骤

配置

配置可执行文件dt-oas-backup.sh，可参照如下配置项示例修改

```
# 在redhat/centos发行版本中需要设置，debian环境中可以将其注释
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib

# 阿里云归档存储和OSS账号信息
export AliCLD_OAS_HOST="alicloud_oas_host"
export AliCLD_OSS_HOST="alicloud_oss_host"
export AliCLD_ACCESS_ID="alicloud_access_id"
export AliCLD_ACCESS_KEY="alicloud_access_key"

# 签名
export PASSPHRASE="gpg_passphrase"

# 备份目录
ROOT="/home/"

# 远程阿里云归档存储vault
DEST="oas+http://oas-backup-101"

# 匹配包含的备份路径
# > 注：INCLIST的路径不要以'/'结束，否则只会上传目录结构而没有真实文件内容
INCLIST=( "/home" )

# 匹配排出的备份路径
EXCLIST=( "/home/xuser" )

# 日志存放目录
LOGDIR="/tmp/"

# duplicity可执行文件路径，一般在python所在路径的bin目录下
DUPLICITY="path_to_python/bin/duplicity"
```

执行

- **全量备份**

```
bash dt-oas-backup.sh --full
```

注：第一次执行备份时，建议执行全量备份。

- **增量备份**

```
bash dt-oas-backup.sh --backup
```

- **校验备份集**

```
bash dt-oas-backup.sh --verify
Verify complete. Check the log file for results:
>> /tmp/duplicity-2015-11-19_10-02.txt
```

- **列出备份文件列表**

```
bash dt-oas-backup.sh --list-current-files
Local and Remote metadata are synchronized, no sync needed.
Last full backup date: Thu Nov 19 10:01:32 2015
Wed Oct 21 10:12:14 2015 .
Tue Oct 20 15:28:23 2015 core
Mon Aug 31 09:44:12 2015 core/__init__.py
Wed Aug 19 09:51:05 2015 core/asyncoro.py
Mon Sep 28 17:20:57 2015 core/job.py
Wed Aug 19 16:29:09 2015 core/policy.py
Wed Aug 19 21:03:34 2015 core/resource.py
Wed Sep 23 10:33:42 2015 core/scan.py
```

- **恢复单个文件**

在源文件保留的前提下，可以进行MD5进行校验

```
bash dt-oas-backup.sh --restore-file core/job.py /tmp/j.py
YOU ARE ABOUT TO...
>> RESTORE: core/job.py
>> TO: /tmp/j.py

Are you sure you want to do that ('yes' to continue)?
yes
Restoring now ...
```

- **设置cron定时任务**

以每天凌晨2点进行备份为例

```
```

```
crontab -e
0 2 * * * bash /usr/local/bin/dt-oas-backup.sh --backup
```